substantive amendments have been made, and no new matter has been added. A marked-up copy of the amended claim is also attached hereto to aid the Examiner in identification of the change.

Claims 10-11 stands rejected under 35 U.S.C. §103(a) as being unpatentable over "Fault-Tolerance for Communicating Multidatabase Transactions" (Kuhn94) in view of U.S. Patent No. 5,734,898 (He).

One of the claimed limitations recited in the independent Claim 10 is that "all coordination servers are **identical** regarding their basic functionality for **distributed** object transaction and process management, and that they, taken together, form a **global operating system**." In other words, the claimed coordination servers (or the coordination kernels) make up a **distributed space**. In a distributed space, the coordination servers must be **identical** in the sense that they implement the same communication protocol concerning the administration of object replicas, distributed transactions, and processes (although they may be different in some other aspects as, for example, they may be implemented in different programming languages on different platforms, etc.).

In contradistinction, the prior art reference Kuhn94 only refers to a "globally shared space" and does not teach or suggest the distributed space. All participants of Kuhn94 on the "globally shared space" have a common view of all communication objects in the space. Thus, according to Kuhn94, the globally shared space is logically seen as one consistent space that can be used simultaneously by all participants. Such a "globally shared space" of Kuhn94 is **not at all** the claimed "global operating system," as recited in Claim 10. In this regard, Claim 10 is substantially different from Kuhn94.

As to the claimed limitations of the updateable objects and the blocking transactional read, it is respectfully emphasized that the presently claimed coordination system is

completely different from He. In particular, the claimed coordination system is a peer-to-peer system, but the system taught in He is a client/server system to the contrary. Accordingly, the server of the presently claimed system runs on each node.

Further, updateable objects of the presently claimed system are coordinated by means of an **optimistic concurrency control**, whereas He uses pessimistic locking as means to control concurrency. It is further emphasized that the presently claimed system does not use explicit locks on objects **at all**. Therefore, the claimed updateable objects have a different meaning when they are compared with those (i.e., "the objects being updated") taught in He. For instance, "the objects being updated" of He do not serve for the communication between different nodes.

According to the presently claimed invention, transactions may write (i.e. physically store data on the disk) an updateable object's data on many distributed nodes—not only on the one server site, as at He. In the present "CoKe" system, different consistency models are supported for objects (e.g., via the selectable distribution strategies such as the basic strategy plus flags as recited in Claim 11).

Therefore, as updateable objects have a specific meaning in the CoKe system, and specific semantics have been developed for the meaning of transactional read updateable objects in CoKe.

For this reason, a logical time stamp is introduced for updateable objects, and the time stamps have other purposes than He's version number:

(1) The logical time stamp is never reset—not even after a cache reload of the object (in the present CoKe system, the logical time stamp is an inseparable part of the updateable object). The read API must specify a logical time stamp (so-called "logical read time stamp") that has nothing in common with the logical time stamp of

the very object in the respective local cache. The semantics of the logical read time stamp are: "read the next value of the object with the condition that the logical read time of this object must be greater than the specified logical read time stamp."

(2) Depending on the consistency model used for the object, the CoKe read-API may or may not return a value that is the most recent value. Only the above-described condition must be met. In contrast, the server in He's system always returns the most recent value of the object.

The advantage of returning a non most-up-to-date value for an object in CoKe is that communication overhead can be avoided—depending on the consistency model used. Ideally, CoKe will return a value from the local cache or a close neighbor's cache, and, if the user decides not to commit the read in its transaction (e.g. he might cancel the read after having received a value), the communication overhead for this read call is avoided at commit time at all.

The CoKe transaction manager uses the logical time stamp for the implementation of optimistic concurrency control: The logical read time stamp is compared at commit time with the actual time stamp of the primary copy of the object. In contrast to the CoKe mechanism, He uses pessimistic concurrency control, which employs (as stated above) locks rather than version numbers.

With respect to the feature of Claim 10 relating to the selectable distribution strategies on which application programs do not depend on (i.e., "distribution strategies are provided for the administration of communication objects, with the application programs not depending on said distribution strategies"), Applicant respectfully submit the following remarks. The Office Action points to Kuhn94, Section 4.1, the 11th paragraph, which states "... maintenance of communication object is separated from the processes executing the

programs that access communication objects." This passage however is to convey the meaning in the context that the administration of the objects takes place in the "CoK"—process, that is, in another process space than that where the application program is running. This means that the implementation of the persistency mechanism is physically separated from the application program context, and thus a developer would have the flexibility to implement and change the persistency model.

However, as recited in Claim 10, the application programs are independent of the distribution strategy that is selected. So, the same APIs can be used for all distribution strategies—the programs need not be syntactically re-written. There exists a common (defined) semantics for the APIs that access communication objects that remains the same independent of whether, for example, a lazy or eager distribution strategy flag is used.

It is noted that to define these common semantics, the logical time stamp of updateable objects plays an important role. The difference does not lie in another usage of the API (e.g. the same read API is used in both cases, returning a value if an only if the logical read timestamp is less than the actual object's logical time stamp), but only concerns communication overhead, performance, availability and the chance that a transaction may commit.

Obviously, there is (semantically) a difference between reliable and unreliable mode concerning reliability behavior, as the first (the reliable mode) allows for automatic recovery of objects and the second (the unreliable mode) does not. However, the way the objects are accessed via the CoKe APIs (read, written etc.) in transactions is syntactically identical to that of the unreliable mode.

Accordingly, the presently claimed subject matter of Claim 10 should not be considered as obvious in view of Kuhn94, even if the reference is taken in combination with

He. For the reasons stated above, Applicant respectfully submits that the claimed limitations of Claim 10 are considered as clearly patentable subject matter in view of the cited prior art references.

Claims 11-18 are also considered allowable at least since these claims depend on Claim 10 that is considered allowable. In addition, these claims are also considered allowable in view of the following remarks.

As to Claim 11, Claim 11 is substantially different from Kuhn94 in that Claim 11 allows the selection of a basic protocol type (e.g. broadcast or passive replication, etc.) and that this basic communication protocol can be parameterized. More specifically, there exist analogous flags for each distribution protocol so that e.g. when selecting a lazy distribution strategy, the basic strategy can be exchanged but the lazy propagation behavior can be relied on. This provides advantages in the implementation of the present CoKe system, namely that only one base protocol needs to be implemented to which (incrementally) new flags can be added. This requires significantly less implementation effort than writing an entire new protocol for each feature.

Claim 12 stands rejected under 35 U.S.C. §103(a) as being unpatentable over Kuhn94 and He in view of "A Distributed and Recoverable Linda Implementation with Prolog&Co" (Kuhn2).

As to Claim 12, Kuhn2 does not properly qualify as a prior art reference under 35 U.S.C. §103(a) (or §102(e)) as asserted in the Office Action, because Kuhn2's publication date (October 2, 1996) is after the priority date (September 30 1996) of this application. (It is also noted that the publication date of Kuhn2 is less than one year before the filing date (September 24, 1997) of the present application.)

Claims 13-18 stand rejected under 35 U.S.C. §103(a) as being unpatentable over Kuhn94, He, and Kuhn2 in view of "Logic Based and Imperative Coordination Languages" (Forst).

As to Claim 13, Forst does not teach or suggest the claimed limitation of Claim 13 that allows both the automatic garbage collection and/or the manual (or explicit) garbage collection, for which operation there exist compatible APIs.

As to Claim 14, Forst teaches independent processes, which are non-transactional processes. Applicant respectfully submits that the idea of the dependent process did not exist at that time of Forst, and as such Forst does not teach or suggest a dependent process. Thus, Forst does not differentiate the process types used therein; only a single notion of the "process" is disclosed in Forst, namely in the sense of an independent process.

The introduction of a remotely running dependent process was not obvious: it required the finding that the transaction model and the process model must be united and how they play together.

As to Claim 16, Applicant respectfully reassert the above remarks with respect to Claim 14 and submits that Forst does not describe distributed subtractions for one of the reasons that the idea of distributed transactions did not exist at the time of Forst.

As to Claim 17, "prepare/1" of Forst calls a process on the result of which the transaction depends. If "prepare/1" fails, the transaction also fails. If "prepare/1" crashes, the called process is not restarted, and the crash consequently would cause the transaction failure. In other words, "prepare/1" is a part of a transaction (a "prepare phase").

In contradistinction, the "on-commitment" (or on-commit) actions as recited in Claim 17 have a different meaning from Forst. The on-commit actions are started if the transaction

is going to definitively perform its commit. That is, the commit of the transaction is not dependent on the on-commit actions. Moreover, the on-commit actions are automatically recovered, if they crash.

The on-commit actions provides significant advantages over prepare/1 in that the on-commit actions allow grouping together the writing of objects and the starting processes in a transaction in an atomic way. That is, if for instance data are communicated into the virtually shared memory space, then a worker can be started reliably by means of an on-commit to operate on these data. In the case of "prepare/1," it could happen that "prepare/1" is started by the objects are finally not written at all because of the "prepare/1" failure. Note that for "prepare/1," the data to be written in the transaction are not yet visible, whereas on-commit sees already these data in the virtual shared memory space.

As to Claim 18, the semantic compensation as described in Forst refers to committed compensatable subtransactions. It serves to guarantee the atomicity of transactions if the isolation property of transactions is relaxed; that is, in case of the enclosing transaction failure, the compensation action is called for the committed subtransaction to compensate its effects in a semantic way.

Claim 18, however, defines a different mechanism, namely the dynamic and programmer controlled repair of transactions on the basis of a single operation, for example, read, write, local subtransaction, etc., after a unsuccessful try to commit the transaction. This means that the programmer may manually cancel (i.e. dynamically back-track) an operation of a transaction that would cause transaction failure. Other operations (i.e. not just the failing ones) can also be cancelled. After a cancellation, the programmer may retry to commit the transaction.

The idea of cancellation is only possible in combination with the invention of hard (trans_commit) and soft commit (trans_try_commit). A cancellation is only possible after when a soft commit has failed. This differentiation of hard and soft commits is not taught or suggested by Forst. It is respectfully noted that a cancellation—contrary to semantic compensation—does not semantically undo the effects of a committed subtransaction but withdraws an operation that was issued in a transaction saying it is not needed henceforth.

For the reasons set forth above, Applicant respectfully submits that the Claims 10-18, pending in this application, are in condition for allowance over the art of record. This amendment is considered to be responsive to all points raised in the Office Action. Accordingly, Applicant respectfully requests reconsideration and withdrawal of the outstanding rejections and earnestly solicits an indication of allowable subject matter. Should the Examiner have any remaining questions or concerns, the Examiner is encouraged to contact the undersigned attorney by telephone to expeditiously resolve such concerns.

Respectfully submitted,

Dated: 1/31/03

Richard J. Streit, Reg. No. 25,765
c/o Ladas & Parry
224 South Michigan Avenue
Chicago, Illinois 60604
(312) 427-1300

DOCKET: CU-1867 PATENT

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Application:	Eva KUHN	]
Serial No:	09/269,485	] GRP ART UNIT: 2151
Filed:	March 29, 1999	] Ex.: ZHEN, Li B.
For:	COORDINATION SYSTEM	]

SPECIFICATION-MARKED UP VERSION

Please amend the Specification page 19, the second paragraph (lines 12-17), as set forth below.

Activities are implicitly synchronized, because if a system depends on the result of another system, it knows which object contains the data, and it can simply perform an access to them. If the data are not yet available, a system wanting to access them will simply **[nedd]** **need** more access time.

Please amend the Specification page 21, the first paragraph (lines 1-18), as set forth below.

The communication objects can be of any type, i.e. they may be assigned a value only once or they may be updateable like variables. Each process can rely on the data read from the global space, because (in case of write-once objects) they will not change, or they are recoverable, respectively. Moreover, the communication objects have a unique identification number, but no global name. These object identification numbers (OID) are suitable exported via the above mentioned name servers, which are realized at application level by means of known database technologies. The communication objects are shared between processes by passing them in arguments. A process having no reference to a communication object will not

gain access to it. The agent maintaining the communication objects prevents processes from obtaining the reference to a communication object by **[triekery]** **trickery**. This gives security, because only authorized processes are granted access to the data.

Please amend the Specification page 25, the third paragraph (lines 17-24), as set forth below.

Regarding **[ist]** **its** general function, the global operating system 24 is shown in Fig. 3 and, in greater detail with regard to transaction control, in greater detail with regard to transaction control, in Figs. 4 to 9 in connection with Figs. 10 to 23 (transaction manager); the process manager is apparent from Fig. 24 in connection with Figs. 25 to 31, and the strategy manager (composed of single managers SMi for the corresponding distribution strategy) from Figs. 32 to 40.

Please amend the Specification page 29, the second paragraph (lines 17-27), as set forth below.

In detail, initially at 44 it is asked whether the local request R is a request for the creation of a new object. If yes, the block 45 "object creation" (see Fig. 6) follows. **[It]** **If** not, next it is asked whether the incoming local request is a request for "object read" (query 46). If yes, in block 47 (see Fig. 7) the command "read object" is executed. If not, as a third query it is checked at 48 whether the local request is a request for "alternative wait" (supervision of objects). If yes, the subprogram "alternative wait", block 49 (see Fig. 8), is called; if not, block 41 in Fig. 4 follows.

Please amend the Specification page 31, the third paragraph (lines 12-16), as set forth below.

It depends on the respective distribution strategy and **[ist] its** flags, whether, it is sufficient for the read request to check the local object structure, or whether communication steps have to be performed which ask other agents for the state and value of the object.

Please amend the Specification page 31, the fourth paragraph (page 31, line 17 to page 32, line 11 line 18), as set forth below.

In detail, with "object read" according to Fig. 7 it is initially tested at 50 whether the process is granted access to the object, and whether the object is of write-once type. **[It] If** the result of this query is negative, at 51 an error message occurs, but if the result, is positive, at 52 a test takes place whether the object state is defined. If it is defined, at 53 the value of the object is returned, and control proceeds to the end of the function (block 41 in Fig. 5). If, however, the object state is not defined, at 54 it checked whether if the reading is blocking (i.e. whether the blocking flag is set), and if not, at 55 an error message occurs; if yes, in step 56 it is then tested whether the request has been issued by the user, which means that there exists no read request structure yet for this read request. If the result is negative, control proceeds to step 41; if the result is positive, according to block 57 a read request structure is created, which is then appended to the object. Then, according to block 58 the strategy manger is called to execute the function that the object is to be read. This step 58 will be illustrated in greater detail below, using Fig. 34.

DOCKET: CU-1867 PATENT

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Application:	Eva KUHN	]
Serial No:	09/269,485	] GRP ART UNIT: 2151
Filed:	March 29, 1999	] Ex.: ZHEN, Li B.
For:	COORDINATION SYSTEM	]

CLAIMS-MARKED UP VERSION

Please amend Claim 14 as set forth below.

14. **(Once Amended)** A system according to claim 13, wherein heterogeneous transactions or subtransactions are distributed to different sites (X,Y,Z) via the coordination **[serves]** **servers** which, taken together, behave as a global operating system.